Introduction to Programming with Python

Computing

Course: CSE 1100 – Introduction to

Lecturer: Lenandlar Singh

Date: December 04, 2025

So Far

- We can get user input
- We can create variables
- We can convert values from one type to another using functions
- We can use conditionals
- We can import new modules with specific functions we need
- We can use lists and loops

What’s Next?- Iteration

- With our random number program we’re only able to let the user guess once before having to restart the program
- But in a real guessing game hopefully the user would be able to keep guessing until they found the answer
- How can we do this?
  - Use iteration

Iteration

- Iteration allows us to write programs that repeat some code over and over again
- The easiest form of iteration is using a while loop
- While loops read a lot like English (like almost everything in Python)
  - While counter is greater than 0, execute some code
  - While userguess does not equal my number, keep asking for a new number

While Loop

- While loops are like if, elif, and else statements in that only the indented code is part of the loop
  - Be careful with indentation
- Like if and elif statements, while loops must check to see if some condition is true
  - While it’s true, execute some code
  - When it’s not true, exit the loop

Simple While Loop Example

- Let's write a while loop that prints the numbers from 1 to 10

```
i = 1              #initialize i 1
while i<=10:       #execute the code in the loop
  print(i)         until i >10
  i = i +1         #increment i
print("all done!")
```

Another While Loop Example

```
count =1
num =3
while count<5:
  print(num*count)
  count = count+1
```

*this prints 3, 6, 9, 12

Variation on Last While Loop

```
num =3
while num<=12:
  print(num)
  num = num+3
```

*this loop also prints 3, 6, 9, 12

In-Class Example

- Adding a while loop to our random program

While loops, cont.

- Now our user can keep guessing until they get the right answer
  - But they can still only play once
- We can add another while loop to allow the user to continue playing
- In-Class Example

Counters

- Often when creating and using while loops we need to keep count of how many times the loop has executed
  - Usually used as the test condition for the while loop
- Ex.

```
count = 1
while count<10
print(“I love programming”)
count = count+1
```

- In this simple example, count is referred to as the *counter*

Counters, cont.

- Counters can be used as part of the body of the while loop
  - The counter can be used for more than counting
- Counters are often given a variable name of a single character
  - i
  - j
  - k
  - a
  - b
  - Etc.

Counters, cont.

- We don’t always need counters

```
count =1
num =3
while count<5:
  print(num*count)
  count = count+1
```

```
num =3
while num<=12:
  print(num)
  num = num+3
```

Functions

- We’ve been using many functions so far
  - print, input,
  - int, float, str
  - random.randint

- So what is a function?
  - Sequence of code which performs a specific task

Functions, cont.

- Functions often take arguments
  - This is what you have to put inside the parentheses
  - For example, raw input “takes in” a string
    - raw_input (“enter your name”)
  - “Takes in” is another way of saying it takes an argument, in this case a string
- Functions often return a value
  - This is what you get back when you “call” the function
  - For example, raw_input returns a string
  - int() returns an integer
  - float() returns a float

Creating New Functions

- We don’t have to rely on only the functions in the Python library
- We can have functions that do almost anything
  - Can be as simple as adding 2 to a number
- Why use functions?
  - Makes our code easier to read
  - Lets us repeat code more efficiently

Creating a New Function

- The syntax for creating a new function is:

```
def functionName(arguments):
```

- Let's define a function called printName that takes in a string:

```
def printName(userName):
```

- Like with if, elif, else, and while blocks, indentation is important
  - The function is the code indented immediately following the function definition

Simple Function

- Let's create a function that does one thing, prints "UGC 1201"

```
def printWord():
    print("UGC 1201")
```

To call this function we simply use its name:

```
printWord()
```

- This function takes in no arguments and returns no values
  - All it does when we call it is print the string "UGC 1201"

In-Class

- Trying the last function in class
- Adding a while loop to it

Functions with Multiple Arguments

- We can have functions with many arguments
  - As many as we want

- Let’s do a math example where we want to do some fancy math

```
def fancyMath(a,b)
        c = a*b+b/ (a+b) +b**a
        print(c)
```

fancyMath(a,b), cont.

- fancyMath takes in two arguments
  - Need to be numbers (either ints or floats)
- When we call the function fancyMath we have to put in two values for a and b

  fancyMath(2,3)

  fancyMath(5.0,1)

  fancyMath(num1, 3)

  fancyMath(num1, num2)
- We can use integers, floats, or variables
  - And any combination thereof
- Remember that the value we pass into the function gets stored in the variables a and b

In-Class

- Using fancyMath

Return Values

- Functions can take in any number of arguments
  - From 0 ->whatever you want
- But functions can only return one value
- To return a value we use the *return* keyword
- We an return numbers, strings, and any values stored in variables

Adding Return to fancyMath

- Right now fancyMath prints the value of c
  - Has the result of our fancy math calculation
- Instead, let's have it return the value so we can do something else with it in the main part of the program

```
def fancyMath(a,b)
        c = a*b+b/ (a+b) +b**a
        return c
```

Returning Values, cont.

```
def fancyMath(a,b)
        c = a*b+b/ (a+b) +b**a
        return c
```

- Now fancyMath will have a value when we call it
- We can assign the value of that function to a variable

```
fancyNumber = fancyMath(3,2.4)
```

Return Values, cont.

- The str() function takes in a number and returns a string
- But so far when using it, we've not assigned the value of that function to a variable before printing
- We can do the same thing with fancyMath

  print fancyMath(2, 3.5)
- The value that fancyMath returns is then printed

Functions, cont.

- If a function returns a value, we can use it as the argument for another function
- fancyMath returns a number
- But if we want to print the result along with other strings, we can convert it to a string so we can concatenate it
- We are using the result of fancyMath(3, 2.3) as the argument for str()

print “Your result is: “ +str(fancyMath(3, 2.3))

In-Class Example

- Just putting it all together

Questions

- From this lecture you should be able to:
  - Use a simple while loop
  - Understand using counters to terminate loops
  - How to define your own function
  - Know what arguments and return values are
  - How to call a function you write
    - How to assign the return value to a variable
  - Understand that functions with return values can be used as arguments of other functions

Homework

- Write a program in python that uses a function called collectVals – the function will collect a list of n numbers from the user via the keyboard and calculates how many numbers are greater than the average of all the numbers

End of Lecture!!!